A Framework for Adaptive Routing

John Y. Ngai
and
Charles L. Seitz

Computer Science Department
California Institute of Technology
5246:TR:87

A Framework for Adaptive Routing

John Y. Ngai
Charles L. Seitz

Computer Science Department
California Institute of Technology

5246:TR:87

July 16, 1987

The research described in this report was sponsored in part by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597, and in part by grants from Intel Scientific Computers and Ametek Computer Research Division.

1 Introduction

Message-passing concurrent computers such as the Caltech Cosmic Cube [1] and its commercial descendents consist of many computing nodes that interact with each other by sending and receiving messages over communication channels between the nodes. For finer grain message-passing concurrent machines, such as the Caltech Mosaic [2], it becomes progressively more difficult to achieve the performance required of the communication network. The bisection capacity of physically realizable communication networks grows more slowly than the expected message traffic, which is at least linear in the number of nodes.

To improve the overall performance of these machines, it is necessary to improve the performance of both the computing nodes and the underlying communication networks. The concept of message-driven processors [3] promises to deliver more raw power to the individual computing nodes in these machines. However, the message latency of the communication networks used in the Caltech Mosaic [2] and second generation cubes have reached a limit of being almost as fast as physically possible for a given wire bisection while using *oblivious* routing [4]. Any improvements on these networks will require an *adaptive* utilization of available network bandwidth to avoid local congestions.

In an adaptive routing scheme, message routes are no longer deterministic, but are continuously perturbed by local message loading. Message packets will tend to follow their optimal routes to destinations in light traffic loading, but adaptively detour to longer but less loaded routes as local *hot spots* spring up. Further, an adaptive routing scheme can potentially support a *hot-spot throughput* which is equal to the total communication bandwidth of the generating node rather than that of a single communication channel [8] as in an oblivious scheme.

Deadlock in the interconnection network is eliminated by the adoption of a packet *exchange* model [5]. Demonstration of freedom from deadlock in deterministic networks is replaced by the need to demonstrate message and node progress in such networks. Reliable concurrent computation requires that the routing algorithm assure progress in the communication network. In this report we present a framework for the discussion of adaptive routing algorithms. It allows us to examine a number of issues fundamental to any message communication network supporting reliable concurrent computation. Specifically, we present a general scheme for constructing progress guaranteed adaptive routing algorithms for arbitrary networks. In section 2 we describe informally the basic motivations behind investigating adaptive routing strategies. In section 3 we state our assumptions and describe

the packet exchange model. We also prove freedom from deadlock and explain the necessity of having progress guarantees. In section 4 we develop a general packet prioritizing scheme for constructing packet delivery guaranteed routing algorithms for arbitrary networks. In section 5 we discuss the problem of guaranteeing progress in each individual node. We develop network utilization disciplines that result in packet injection guaranteed policies.

The resolution of these fundamental issues provides a foundation upon which one can construct a variety of different adaptive routing strategies. It also provides the basic skeleton upon which one can explore various extensions in the adaptive framework. The present report concentrates in laying down this foundation and in demonstrating the feasibility of the adaptive routing approach. The performance analyses and simulation results of the various strategies and extensions of the basic framework will be the subject and report of future studies.

2 Motivation

In this section, we describe informally the basic ideas and motivations behind the adaptive routing approach in order to place the content of this report in perspective. For illustration purposes, we shall confine ourselves to the two dimensional mesh connected network topology. The concepts can easily be extended to other regular communication structures.

We start by observing that on a $2D$ mesh, if we want to forward a message packet to a node distance d_x away along the x direction and distance d_y away along the y direction, there are a total of $C_{d_x}^{d_x+d_y} = C_{d_y}^{d_x+d_y}$ different shortest distance paths. At places where neither d_x nor d_y is zero, sending the packet out along either the x or the y direction towards its destination are equally *profitable*. Indeed, depending on a packet's current position relative to its destination, it may profitably be routed out in one of the eight possible output channel combinations: $+X$, $-X$, $+Y$, $-Y$, $+X \vee +Y$, $+X \vee -Y$, $-X \vee +Y$ and $-X \vee -Y$. An adaptive control can take advantage of the extra flexibility of some packets being able to route profitably in more than one direction. This is in contrast to existing *oblivious* routing strategies where the entire route of a message is determined solely by its source and destination nodes. For example, consider the situation depicted in figure 1, where we have a node with three incoming message packets and one newly injected message packet. The four packets request respectively, the output channels $+X$ only, $-X$ only, $-Y$ only and $+X \vee +Y$. If we follow the oblivious deadlock free strategy of always routing along the X direction first, then only one packet can get through the $+X$ channel, and the other one

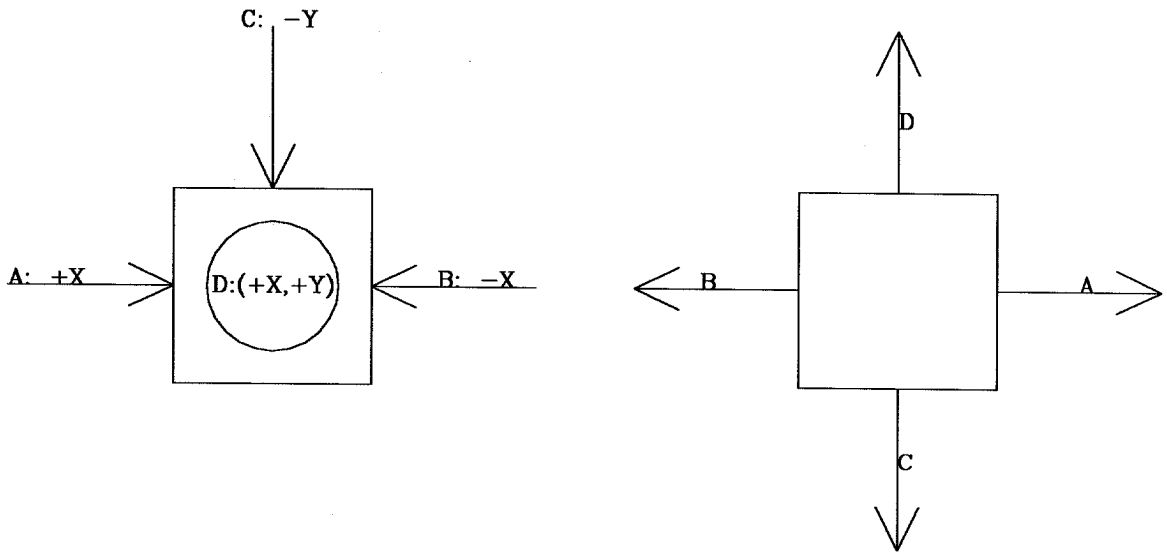

Figure 1: A Simple Adaptive Example

would be blocked. Under an adaptive control, all four packets can be forwarded profitably. An adaptive strategy allows a node to perform a locally optimized decision based on the packets present.

The fundamental characteristic of a *distributed* adaptive routing control is that it forwards any packet sent into the node according to a decision policy based entirely on locally available information. The local decision policy adopted may assume a variety of different forms. As a concrete example, a *dispersive* policy suggested in [5] would forward any incoming packets irrespective of whether it can make any progress. If all four packets request the $+X$ channel, only one would get through profitably, while the other three are all sent to neighboring nodes which are more distant from the packets' respective destinations. The rationale behind this dispersive approach is that by sending the competing packets to adjacent nodes, we manage to broaden up the locally congested bottleneck. The performance analyses and simulation results of these various strategies and other extensions will be the subject and report of future studies.

3 A Simple Adaptive Model

In this section, we describe a simple model for the discussion of adaptive routing strategies in multicomputer networks. We deliberately include just enough structure to provide a context for the discussion of a number of fundamental issues in distributed systems. The following definitions develop the notation for describing multicomputer networks and message routing assignments.

Definition 1 A *Multicomputer Network, M,* is a connected undirected graph, $M = G(N, C)$. The vertices of the graph, N, represent the set of computing nodes. The edges of the graph, C, represent the set of bidirectional communication channels.

Definition 2 Let $n_i \in N$ be a node, the set $C_i \subseteq C$ is the set of bidirectional channels connecting n_i to its neighbors in M. However, for the convenience of presentation, we shall occasionally need to distinguish between the input and output direction of a communication channel. We shall use the notations I_i and O_i to refer specifically the set of *input* and *output* channels of n_i when needed.

Definition 3 For each destination node n_j, the set $C_{ij} \subseteq O_i$ is the set of output channels connected to n_i, whereby forwarding a message packet destined to n_j from n_i through a channel $c_k \in C_{ij}$ will reduce the packet's distance from its destination.

Definition 4 For each node $n_i \in N$, the *Routing Relation* $R_i = \{(n_j, c_k) : n_j \in N - \{n_i\}, c_k \in C_{ij}\}$ defines for each possible destination node $n_j \in N$, its corresponding message forwarding channel set C_{ij}.

We assume the following:

- A message packet arriving at its destination node is consumed.
- A node can generate message packets destined to any other node.
- Node n_i has a total of $b_i \geq |C_i|$ internal message packet buffers.
- Nodes can produce packets at any rate subject to the constraint of available buffer space. (Source Queued)

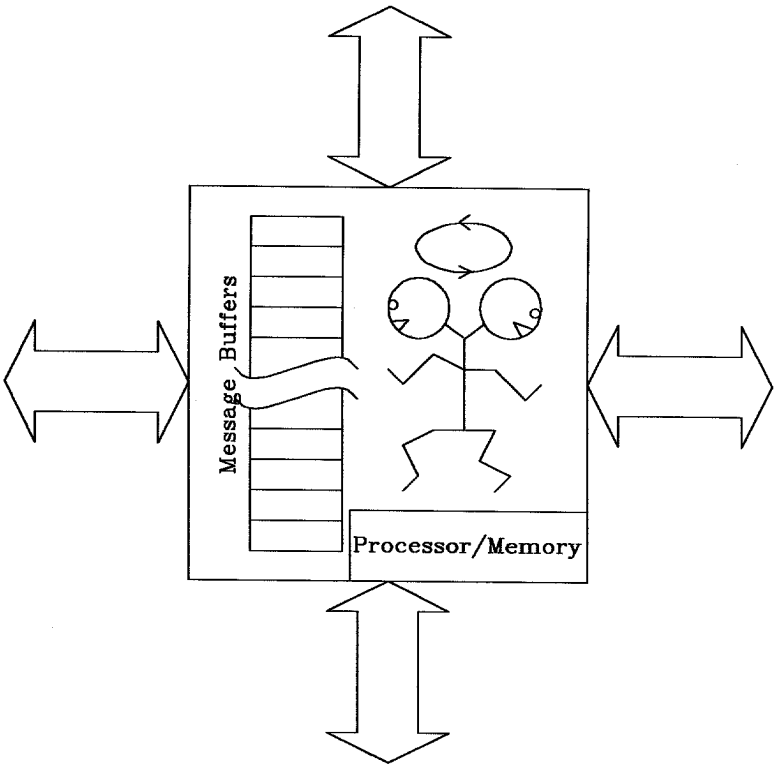

Figure 2: Structure of a node

- Node n_i in the network has complete knowledge of its own routing relation R_i. Such could be accomplished by storing routing tables or by having a well-defined procedure to calculate elements of R_i on the fly. Such procedures are particularly simple in highly regular network topologies.

Figure 2 presents a pictorial view of the structure of a node in a multicomputer network. Conceptually, a node can be partitioned into a computation subsystem and a communication subsystem. The computation subsystem consists of conventional processor and memory modules whose internal structures do not concern us. For our purpose, it serves as the producer and consumer of the messages routed by the communication subsystem of the node. Internally, the communication proper consists of an adaptive control and a small finite number of message packet buffers. Routing decisions are made by the adaptive control based entirely on locally available information.

3.1 The Exchange Model

If there is any advantage in making adaptive routing decisions over making oblivious ones, it must come from the flexibility to exploit alternative routes. Such flexibility must be introduced in ways that still guarantee freedom from deadlock. The concept of *virtual channels* [6] provides a way to develop deadlock-free routing algorithms in arbitrary direct networks. While it may be possible in certain special cases to allow for multiple alternative routes using virtual channels, the task becomes progressively more difficult in more complex routing schemes and network topologies. The concept of a *structured buffer pool* [7] developed for store-and-forward computer communication networks provides a general approach to introduce alternate deadlock-free routes into an arbitrary packet switching network. However, these schemes in general require a buffer pool that grows with the network size, and hence are not feasible in fine grain multicomputer communication networks where resources at the node level are finite and scarce and the network is arbitrarily extensible.

In this paper, we consider networks that employ *exchange* [5] rather than the *block until clear then forward* mechanism used in *wormhole* [4] routing or the *packet acknowledge* mechanism used in *store-and-forward* [7] routing, as the basic node to node message transfer protocol. The exchange model avoids the necessity of having to block, holding a packet and waiting until the requested resource is available and acknowledged, by having the two communicating parties agree to preempt the requested resources if necessary. In the event where both parties are full, the preemption is accomplished through an exchange. Packet transfer is guaranteed locally between every pair of adjacent nodes acting as partners in the exchange operations. This is summarized in the following additional assumptions:

- Packet transfer between adjacent nodes is accomplished through an exchange operation.
- When a node requests a packet transfer, the receiving node will acknowledge the transfer within a *fixed bounded finite* amount of time as discussed below.
- The receiving node simply accepts the incoming packet if it has an available empty buffer to hold the packet, and the receiving node is said to have exchanged a *null* packet with the sending node. In this case the exchange operation is guaranteed to complete within a fixed bounded finite period of time.
- If all the buffers at the receiving node are occupied, the exchange proceeds by having the receiving node *preempt* one of its filled buffers which is not pending for output, by

sending the preempted packet to the sending node. Again, the exchange operation is guaranteed to complete within a fixed bounded finite period of time in this case.

As long as the conditions enumerated in the above assumptions are satisfied, the exchange model would allow us to exploit arbitrary alternate routes without having to worry about potential communication deadlock. This fact is established in the following theorem:

Theorem 1 A multicomputer network, M, employing the exchange protocol of node to node message transfer is deadlock free, i.e. every node guarantees that all input requests will be satisfied within a bounded finite period of time.

Proof. Following the preceding discussion, an incoming request for packet exchange will be completed within a bounded finite period as long as there exists within the requested node, an idle buffer not participating in exchange with any other input channels. Since for each node n_i, $b_i \geq |C_i|$, such a condition is guaranteed. **Q.E.D.**

Since, under our assumptions, an exchange operations will always complete in a bounded finite period, we shall henceforth regard the exchange operation as a single *atomic action* of our networks. The absence of communication deadlock established in the above theorem makes no assumption on the relative timing of the exchange operations between the different communication channels of the same node. Its correctness depends only on the boundedness of the exchange operation duration. In order to simplify our subsequent discussion, we shall assume that the exchange operations across all communication channels of a node are *synchronized.* Specifically, we assume that the exchange operations are organized in discrete rounds henceforth referred to as routing cycles. The synchronized model assumption is not necessary but helps to simplify our subsequent exposition.

3.2 Potential Lack of Progress

The exchange model eliminates the possibility of deadlock by guaranteeing packet transfer at the node to node level. Packet transfer at the source to destination level, however, is not immediate. Consider the situation depicted in figure 3 of a ring network of seven nodes, each with two internal message packet buffers. Each node has initially in it two packets destined to a node distance two from it in the clockwise direction. Suppose each node employs the following deterministic assignment of packet to channel: whenever the two incoming packets both request the same outgoing channel, the packet from the clockwise neighbor

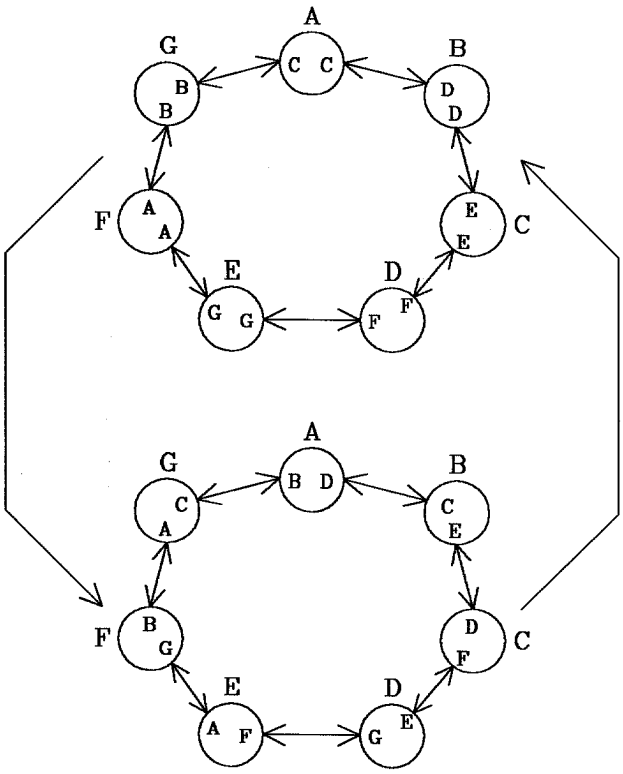

Figure 3: Lack of Message Progress

always wins. The sequence of routing scenarios is also shown in figure 3. Notice that after two routing cycles, we are back to where we have started! This simple example illustrates that it is possible for packets to be forever denied getting delivered to their destinations. Clearly additional measures have to be introduced in order to guarantee delivery of packets, in order words, to guarantee progress in the network. We distinguish several different types of progress guarantees that should be present in a multicomputer communication network.

First, every individual packet in transit inside the communication network should be guaranteed delivery within a bounded finite period. A weaker form guarantees *some* packets in transit inside the network will be delivered within a bounded finite period. In this weaker form, we are assured of some global progress inside the network, but can make no corresponding statement about each and every packet.

The second type of progress guarantee concerns the initial injection of packets into the communication network. Consider the *pathological* situation depicted in figure 4, where a node situated at the cross point of continuous heavy traffic can be denied sending packets into the network for an unbounded amount of time. Ideally, every node that has a packet queued for entry into the network should be guaranteed access into the network within a

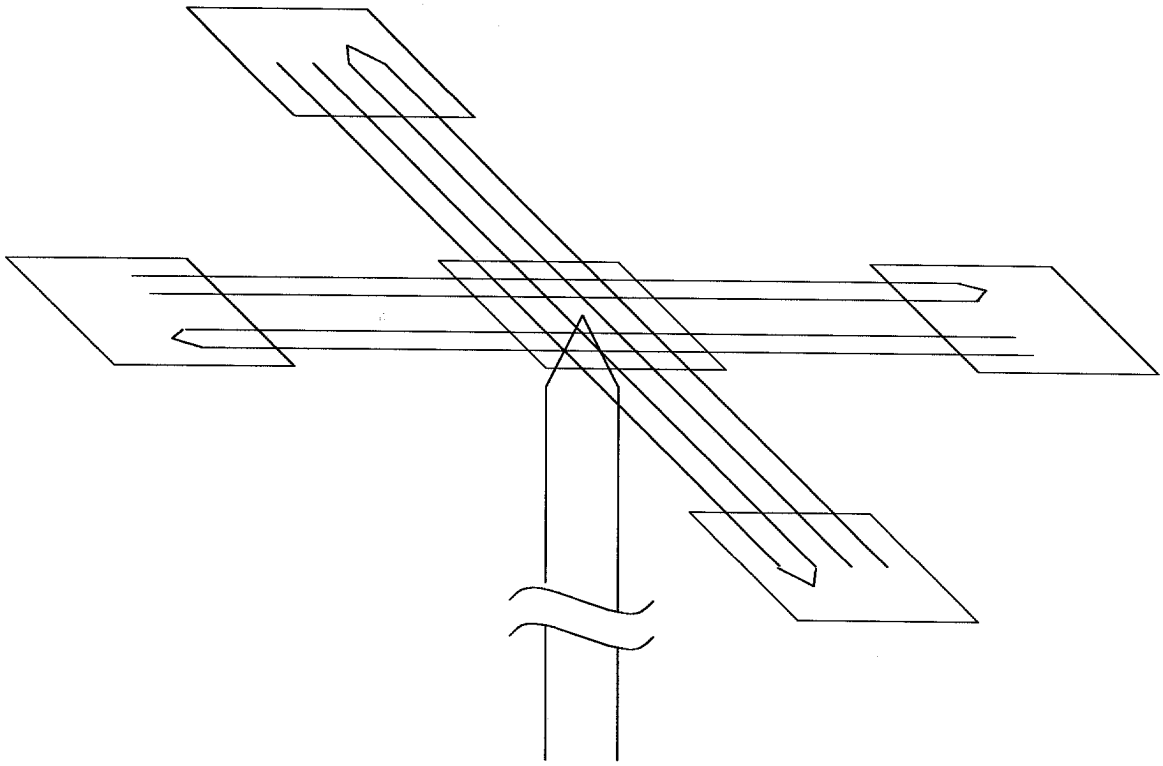

Figure 4: Lack of Node Progress

bounded finite period.

There are a number of other related considerations besides the above two types of progress guarantees. Depending on the intended programming model and semantics to be supported by the multicomputer network, it may be necessary to guarantee preservation of message order between any pair of communicating nodes [4]. However, this requirement can be handled more naturally at a higher level of abstraction than we have discussed here. A closely related issue concerns the transmission of multi-packet messages. In an adaptive routing scheme, the different packets of a multi-packet message may be forwarded along different routes and do not necessarily arrive at the destination node all at the same time. If a message can only be consumed at the destination node after the whole message has been received, then *reassembly deadlock* can arise where packets of different messages fill up all the available buffering space of a node and prevent any of the partially delivered messages from reassembly. Here, we shall assume that message reassembly is accomplished in the node memory, which is normally much larger than the relatively small number of buffers available inside the communication proper. However it should be pointed out that an extension of the exchange model and packet delivery guaranteeing scheme to be discussed

can be employed by the node processor to guarantee freedom from message reassembly deadlock within the node processor memory. However, this issue properly belongs to a higher level of discussion and shall not be pursued further in this expository report.

In summary, we shall focus in describing basic schemes to assure individual packet injection and delivery in networks that employ adaptive routing control under the packet exchange model.

4 Packet Delivery Guarantees

The preceding discussion establishes the need to assure individual packet injection and delivery. In this section, we address the problem of devising general schemes for the assurance of individual packet delivery. General schemes for the assurance of individual packet injection will be discussed in the next section.

In general, given a fixed network topology and the set of routing relations in each node of the network, there could conceivably be a number of different ways to assure eventual packet delivery. For example, one may approach the problem by picking a packet to channel assignment scheme which exploits the underlying network topology in assuring packet delivery. For instance, in our previous ring network, by agreeing only to send packets out in the clockwise direction on a first-in-first-out basis, a restricted form of packet to channel assignment, we can actually assure individual packet delivery. However, such *ad hoc* schemes are extremely difficult to devise and reason about when one considers more complicated topologies. In this section, we present the simple idea of a packet priority scheme as a general method for assuring individual packet delivery. For the following discussion, we assume the synchronized exchange model.

4.1 Static Environment

We start with a *static* finite system where no new packet is to be injected into the network until all present packets are delivered to their respective destinations. In this case, there are only a finite number of packets inside the network. These packets compete among themselves if necessary to gain access to the communication channels that bring them closer to their respective destinations. To each packet, we now assign a fixed *priority* chosen from a *linear order* and in such a way that no two packets have the same priority. For example, if no two packets inside the network are generated from the same source node, then the *source node*

identifier[1] of each packet can be used as the priority of that packet, for example:

$$P_1 > P_2 \Longleftrightarrow N_1 < N_2 \tag{1}$$

where P is the priority of a packet and N its source node identifier. Resolution of channel access conflicts are now based on the priorities of the competing packets, so that the *highest* priority packet among the competitors is always the winner. We now show that this is enough to guarantee eventual delivery of every single packet inside the network.

Lemma 2 Any packet to channel assignment strategy that observes the packet priority ordering defined in equation (1) guarantees eventual delivery of all the packets inside a static finite system.

Proof. During any routing cycle, the packet with the highest priority will always win in its competition and hence will eventually reach its destination and be removed from the network, thus reducing the total number of packets. Upon its delivery the packet with the next highest priority then takes its place and the above argument can be applied again. Since we have only a finite number of packets inside the network, we are guaranteed that eventually all the packets will be delivered. **Q.E.D.**

The fixed priority ordering defined in equation (1) assumes that each node has injected at most one packet into the network in our static environment. This is unnecessarily restrictive, and we present a different priority assignment scheme that removes this restriction.[2] This alternate scheme is also interesting in its own right in that it allows us to look at the routing actions from a different point of view which leads to a *weaker* form of global network progress guarantee for a *dynamic* environment.

The process of forwarding a packet towards its destination can be viewed as a sequence of actions performed to reduce the packet's *distance from destination.* In a channel access conflict, the winner will be routed along a profitable channel, and hence decreases its distance from destination. The losers, depending on whether they are routed away along the remaining unprofitable channels, may or may not increase their distance from destination. Ideally one would like to have strict monotonic decrease of distance to destination for each packet inside the network. This being impossible in our exchange model, the alternative is

[1] Assuming of course that the node ID's form a linear order

[2] Another way to achieve the same objective is to have a finer priority assignment that distinguishes among the various packets from the same source node. This is feasible since a node can only inject a bounded number of packets per cycle.

to ensure monotonic decrease over a sequence of exchanges involving multiple number of packets. This can be achieved by giving higher priority to packets with shorter distances from destination over those with longer distances. Hence, we are motivated to define our distance priority ordering as follows:

$$P_1 > P_2 \Longleftrightarrow D_1 < D_2 \tag{2}$$

where D is a packet's distance from destination. We now show that this alternate form also guarantees eventual individual packet delivery in a static environment.

Lemma 3 Any packet to channel assignment strategy that observes the priority ordering defined in equation (2) guarantees eventual delivery of each individual packet inside a static finite network.

Proof. At the beginning of any routing cycle, let $d > 0$ be the smallest packet distance from destination. During this cycle, a packet with distance d competes with other packets for the use of channel(s) leading to its destination. If it wins the competition, the smallest distance from destination will be reduced to $d-1$. It it loses, it must be losing to another packet also of distance d away from its destination according to the priority policy. Hence, the smallest distance from destination again will be reduced to $d-1$. Therefore d must eventually be reduced to zero, in which case a successful packet delivery occurs and the above argument can be applied again. The finiteness in the total number of packets in our static network then guarantees eventual delivery of all the packets. **Q.E.D.**

In some networks, such as the k-ary-n-cubes or meshes, the address of the destination node of a packet is completely specified by the relative distances the packet has to travel in each dimension. Hence, the relative distance from destination in such networks can be used to determine both the destination node and the packet priority according to our last priority assignment scheme.

It is interesting to note that in the above proof, if we relax the restriction of a static message environment and allow new packets to be continuously injected into our network if feasible, the conclusion of the occurrence of packet deliveries remains valid and allows us to establish *livelock freedom* in our network. In other words, the packet priority ordering defined in (2) alone suffices to guarantee global progress in a message network operating under a dynamic environment. However, no corresponding statement can be made concerning each individual packet.

4.2 Dynamic Environment

The above lemmas give us almost what we want. However, in a dynamic environment where new packets are continuously generated, it is necessary to modify the above priority schemes so that the arguments employed in the proofs of the above lemmas remain valid. For example, to extend the fixed priority ordering defined in equation (1), we need to:

1. Assign a unique and distinct packet priority to every packet currently in transit inside the network.

2. Once a packet has been assigned a particular priority, the message system can only assign higher priorities to at most a bounded finite number of packets during the life time of this packet.

Condition 1 is used to maintain the linear ordering for the priorities among all potential competitors. It can simply be satisfied by having the priority values be chosen from members of a linear ordering of infinite cardinality. Condition 2 is used to replace the finiteness property of its static counterpart. For example, a legitimate extension $P = (B, N)$ of the fixed priority ordering can be defined as follow:

$$(B_1, N_1) > (B_2, N_2) \Longleftrightarrow (B_1 < B_2) \vee ((B_1 = B_2) \wedge (N_1 < N_2)) \tag{3}$$

where B is a packet's *birthdate.*

Similarly, to extend the distance priority ordering described in the preceding discussion, we observe that it has the defect of allowing newly injected packets which have shorter distances from destinations to defeat older packets which have longer distances from destinations. It is not hard to imagine the *pathetic* case where a packet is defeated indefinitely in channel access competitions by a steady stream of newly injected packets. This situation can be rectified by assigning lower priorities to the younger packets as follows:

$$(A_1, D_1) > (A_2, D_2) \Longleftrightarrow (A_1 > A_2) \vee ((A_1 = A_2) \wedge (D_1 < D_2)) \tag{4}$$

where A is a packet's *age*, that is, the number of routing cycles elapsed since the injection of the packet. This new priority ordering is similar to the priority ordering defined in equation (3) except that we have replaced packet *birthdates* with packet *ages.* We now observe that if we redefine the priority ordering in (3) in terms of packet ages as follows:

$$(A_1, N_1) > (A_2, N_2) \Longleftrightarrow (A_1 > A_2) \vee ((A_1 = A_2) \wedge (N_1 < N_2)) \tag{5}$$

we can still satisfy both conditions 1 and 2 since all the packets of a network ages together. The difference is that, with age, the priority assignment of a packet is no longer fixed but has to be updated after every routing cycle.

We now prove that the priority orderings defined in equations (4) and (5) guarantee eventual delivery of every individual packet in a dynamic operating environment. This is summarized in the following theorem:

Theorem 4 Any packet to channel assignment scheme that observes the priority assignment ordering defined either as in equation (4) or as in equation (5) during channel access competitions guarantees eventual delivery of every individual packet inside a finite network.

Proof. During any routing cycle, let P denote the set of packets currently in transit inside the network and $S = \{p_i \in P | A_i \geq A_j \; \forall p_j \in M\}$ denote the set of oldest age packets. We observe that the packets in S form a static population satisfying the premises of either Lemma 2 or Lemma 3, and hence every packet in S will eventually be delivered. As $|S|$ is reduced to 0, packets of the next oldest age group then become members of S, and the above argument can be applied again. Since with either of the priority orderings defined over a finite network, each packet injected can only have a bounded finite number of packets with age older than itself, we are guaranteed that eventually every individual packet will be delivered. **Q.E.D.**

The above theorem provides us with well defined packet priority schemes which can be employed to assure packet delivery for individual packets and hence establishes *lockout freeness* in our networks. There is however an implementation issue that needs to be settled. In a continuously operating message system, we would like to be able to put an upper bound on the maximum age of any packet ever routed in the network. This is necessary in order to allow for a bounded finite implementation of packet priorities. We now show that such a bound exists as long as packets arriving at their destinations are consumed.

Theorem 5 Let M be a message network that satisfies the consumption assumption and observes the priority orderings defined above. Let n be the total number of nodes in M; b be the maximum number of internal buffers of each node and d be the diameter of M, then no packet ever routed in the network can have an age $> bnd$.

Proof. Observe that when a packet p_i is first injected into the network, there can be at most $bn-1$ older packets already around in the network. Since after at most every d cycles,

there must be at least one successful packet delivery, therefore after at most $(bn-1)d$ cycles, p_i must becomes the oldest packet in the network if it has not already been delivered. In this case p_i is guaranteed to be delivered to its destination and be consumed by the end of the bnd-th cycle. **Q.E.D.**

Armed with the above theorems, we can be sure that every packet sent will be delivered to its destinations and within bounded time under the consumption assumption. There remains, however, the problem of injecting a packet into the routing network in the first place. Notice that for a node located inside a region of heavy message traffic, new packets generated by it have to compete for channel access with packets already in the network. An immediate naive response would be to simply treat the newly generated packet just like any other packet and have it compete for channel access according to the assigned priority ordering. In the next section, we shall see that the problem is actually deeper than it appears.

5 Packet Injection Guarantees

One major resource limitation that reveals itself when the node *grain size* shrinks is that of limited finite buffering capability per node. In addition to requiring a steady state dynamic equilibrium between incoming and outgoing packets, each node also becomes less tolerant of transient fluctuations. In a deterministic blocking scheme, transient fluctuation of message traffic is handled by denying it from happening, i.e., by blocking. In our adaptive strategy, it is handled by maintaining strict balance between incoming and outgoing packets. This is possible only because we are willing to route packets away from their respective destinations if necessary. However, when new packets are injected into the network from a node, this delicate balance is violated locally. The opposite happens when a packet arrives at its destination, where it is consumed. This violation stems from the unavoidable decoupling of message generation and message consumption in both the temporal and spatial dimensions. As message packets are generated at each node and injected into the network, the input/output dynamic balance is violated locally. Consider the situation where a node n_i that has b_i internal message packet buffers. Suppose this node is located in the middle of heavy message traffic, so that all its input and output channels are continuously busy for a long period. Any message packets generated from the node will be queued up in the internal message packet buffers, and compete with the passing message traffic for access to the output channels. Our packet prioritizing schemes assure that the queued up packets

will eventually gain access to the output channels and be delivered to their destinations. However, if passing message traffic remains heavy for a prolonged period, in the worst case, the node can run out of internal buffers after it has injected b_i packets into the network. Further injection of message packets must first wait for the internal buffers to free up. Conservation of packets requires the internal packet buffers to remain filled unless:

1. The node receives some input message packets destined to itself, whereby it consumes them and frees up their corresponding buffer spaces.

2. Some input channels becomes idle, i.e. receives *null* packets during their exchange operations, whence, the node frees up some of its internal buffer spaces.

In the following subsections, we will describe two different approaches to assure message injections at every node, each corresponds to guaranteeing the eventual occurrence of one of the above two conditions.

5.1 Buffer Token Recirculation

In this section, we describe a method whereby a node can guarantee the emptying of its internal message packet buffers by consuming input packets destined to itself. In general, a node cannot depend on the other nodes to send message packets to itself since it may not be the destination of messages generated by any other nodes in the network. When a node sends a packet to a different node in the network, it has the effect of migrating a *hole* from the source node internal buffer to the destination node internal buffer. In our scheme, we require that the destination node to return this *hole* back to its *owner* by sending a round trip packet back to the source node. In essence, the internal message packet buffers represent reusable tokens which are recirculating between the senders and receivers of messages.

Specifically, let b_i be the total number of internal packet buffers of the node n_i. Our scheme requires that $b_i > |C_i|$. In order words, we require extra buffers in addition to the ones used in the normal exchange operations. Among these extra buffers of each node, we assign a fraction of them as privately reserved buffers used exclusively for packet injections from the local host. We achieve this by tagging these reserved buffers with *private* tokens belonging exclusively to the node n_i. In contrast, all the other remaining internal buffers of the node n_i are tagged with *public* tokens that belong to the whole network.

Our packet injection policy works as follows. When a node has a packet pending injec-

tion, it employs the *greedy* strategy of always placing the packet in an empty internal buffer holding a public token if one is available. When all buffers holding public tokens are occupied, it would then place the packet in a empty buffer holding its own private token if one is available; otherwise it waits until the next available empty buffer appears. Once a packet, including a null packet, is placed into an internal buffer, it picks up the token currently held by the buffer and carries it along while the packet is routed towards its destination. At the destination where the packet is consumed, the carried token is handed back to the buffer currently storing the packet. If the token is a private token belonging to some node other than the receiver, the receiving node is obliged to return it to its owner by sending a reply packet carrying this token back to its owner. In this way the recirculating token is brought back to its owner by the reply packet, and assures the eventual occurrence of packets destined to a node if that node has a packet injection pending due to the absence of empty internal buffers. Hence, we can guarantee eventual packet injection for each pending node. Notice that the validity of our packet injection guarantee depends on our ability to guarantee eventual deliveries for injected packet in our network. This dependence is to occur again in our next scheme.

The above packet injection policy and token recirculation discipline allow us to assure eventual packet injection at each pending node of the network. However the scheme suffers from the inherent inefficiency of having to route reply packets in order to carry the empty buffers (private tokens) back to their owners. Conceivably, it appears more efficient for a pending node to seek available empty buffers starting from its own vicinity. Further, the population of round trip packets which carry no useful information increases when network message traffic is heavy, which is precisely the moment when network resources are scarce. In summary, the token recirculation scheme leaves much to be desired.

5.2 Packet Injection Control

The buffer token recirculation scheme described in the previous section employs a *passive* mechanism whereby each individual node maintains its own capability to inject packets into the network defensively. This is achieved by reserving private buffers which can be occupied only by message packets generated by itself, or by packets involved in exchange of its injected messages. The round trip packets then preempt these private buffers rendering them ready for further packet injections. On the contrary, our second approach employs a distributed mechanism whereby each node *actively* defend its own right to inject packets into the network. The same idea is in fact, involved in the realization of a distributed clock

network that provides the physical basis of an arbitrarily extensible synchronized network which justifies the feasibility of our synchronized exchange model. This realization is the subject of a separate report in preparation.

Specifically, when all internal buffers are full, a node has to wait for some input channels to become idle before it can inject another packet. Our scheme is then a distributed mechanism that guarantees the eventual occurrence of such idle input channels.

Ideally, whenever a node has a packet queued up for injection, it should be allowed to do so. We shall adopt the view point that if a node has no packet to inject, we regard it as if the node has a *null* packet ready for injection into the network. That is, by convention, *during each routing cycle, every node has a packet ready to inject.* Our scheme is to introduce *local synchronization* between each node and its neighboring nodes such that the total number of packets injected by a node after each routing cycle cannot differ by more than a fixed positive constant from those of its neighbors. The null packet convention is required to prevent quiescent nodes which do not have any packet to inject from blocking injections in the active nodes. For explanatory purposes, we describe below the simplier case where each node in the network is allowed to inject at most one packet per routing cycle. The modifications required in order to allow for multiple packet injections per cycle is straightforward.

We now describe the local synchronization protocol and packet injection policy in a general conceptual framework. For our protocol, we assume that each node explicitly maintains records of the total number of packet injections for each of its neighbors measured *relative to that of its own.* The information required to update these records in each node is assumed to be piggybacked on the packets exchanged with its neighbors during each routing cycle and forms an integral part of the packet headers. The information exchanged with a node's neighbors during each routing cycle is the total number of packet injections by the node during the current cycle. Records for neighbors' injections are maintained in the form of relative differences because the absolute value in the number of total packet injections in a continuously operating environment can grow without bound. We now describe the protocol in detail:

0. Immediately after system initialization, each node has the records of the relative differences of packet injections for each of its neighbors be initialized to 0.

1. At the beginning of each cycle, each node examines the recorded relative differences in total number of message injections for each of its neighbors and computes their minimum. A node is allowed to inject its queued packet only if the computed min-

imum is greater than $-k$, i.e., the node is less than k packet injections ahead of its minimum neighbor. Recall that by convention a node always has a packet queued up for injection.

2. If a node is not allowed to inject its queued packet, it just performs its normal packet to channel routing assignments and then goes to step 3. If a node is allowed to inject, it examines its next queued packet. Null packets are always injected by convention, whereas real packets are injected only if the node finds an available empty buffer for it. A node then performs its normal packet to channel assignments and then continues to step 3.

3. Each node computes its own number of packet injections during the current cycle according to the results obtained from steps 2, where null packet injections are counted just like real injections. This number is then inserted into the outgoing packet headers. These packets are then forwarded in exchange with incoming packets from the neighboring nodes.

4. At the end of the routing cycle, each node updates the corresponding record of each of its neighbors by adding the corresponding numbers extracted from the incoming packet headers and subtract from each of them the number it computed in 3. Each node is now ready for the next routing cycle starting again at step 1.

We now show that the above sequence of actions guarantees that after each routing cycle, the difference in the total number of packet injections between neighboring nodes is at most k. This is summarized in the next lemma:

Lemma 6 The local synchronization protocol and packet injection policy described above guarantees that after each routing cycle, the difference in the total number of packet injections between neighboring nodes is at most $k > 0$.

Proof. Let c denote the number of routing cycles elapsed since initialization. The proof is by induction on c. After $c = 0$, i.e., immediately after initialization, the differences are exactly 0 in every node, and hence the statement of the lemma is valid. We now assume that the statement is valid after $c = m$, i.e., the magnitude of the differences i in the total number of packet injections between neighboring nodes is at most k. We now focus on a particular node n. At the beginning of the $(m+1)$-st routing cycle, n has complete knowledge of the differences in packet injections of its neighbors relative to itself up to and including the m-th cycle. By convention, neighbors which are ahead of n in injections

have relative differences $i > 0$, while those lagging behind n have their relative differences $i < 0$. In general, n would have some neighbors having differences zero, some positive, and some negative. If n has some critical neighbors with $i = -k$, then according to the synchronization protocol, n is denied injection in this cycle. Thus the change in i of n's neighbors can only be 0 or $+1$ during this cycle. Therefore, after this cycle, the differences in the total number of packet injections between n and its critical neighbors are still at most k. For those neighbors with $-k < i < k$, no matter what n and its neighbors do in this cycle, their differences are guaranteed to be at most k. For those neighbors with $i = k$, n is itself a critical neighbor of them, and the above argument for n's critical neighbors can be applied to guarantee a difference of at most k. Hence, after the $(m+1)$-st routing cycle, the difference in total number of packet injections between any pair of neighboring nodes remains at most k, and hence the validity of the lemma. **Q.E.D.**

The above lemma provides us with a synchronization protocol that guarantees a constant maximum difference in the total number of packet injections across neighboring nodes in our network. This protocol establishes a distributed discipline for each of the nodes inside a network to cooperate in order to assure the eventual occurrence of idle input channels whenever a node has a real packet waiting for injection which is permitted by the protocol. To accomplish this, we now show that all we need is eventual delivery of the packets already injected into the network. This is summarized in the following lemma:

Lemma 7 A node which has a packet waiting for injection that is permissible under the local synchronization protocol and packet injection policy will eventually be able to inject.

Proof. First, we observe that by our convention, if the pending packet is a null packet, the node is able to inject immediately, so that the lemma is valid vacuously. We now proceed to establish the validity for real packets.

Suppose to the contrary, that a particular node n inside the network is blocked from injecting its queued packets indefinitely because it cannot find an idle input channel and all of its internal buffers are occupied. Our synchronization protocol and packet injection policy then dictates its neighbors to also be blocked indefinitely from injecting their queued packets into the network. These in turns indefinitely blocks their neighbors and so. Thus, given a finite network, all the nodes inside the network are eventually blocked from any further packet injection, and eventually no new packet can enter the network. Since the message system guarantees eventual delivery for the packets already inside the network, eventually the network will be void of messages, whence, every input channel to n becomes idle, which

in turn enables the node n to start injecting its queued packet into the network. This contradicts the original indefinite blocking assumption of n. Hence, we conclude that if a node has a packet injection pending which is permitted by the protocol, it will eventually be able to inject. **Q.E.D.**

We are now ready to show that by following the above protocol and injection policy, we can be assured of progress in every individual node of our network. Specifically, let M be a network and let T_i denote the total number of packet injections from node $n_i \in M$ since initialization. We prove that T_i is strictly increasing over time. This is summarized in the following theorem:

Theorem 8 A local synchronization protocol and packet injection policy that guarantee at most a constant difference $k > 0$ in the total number of packet injections across neighboring nodes, together with a message system that guarantees eventual delivery of each individual packet inside a finite network, guarantees progress in each individual node, i.e., the total number of packet injections for each node is strictly increasing over time.

Proof. During any routing cycle, let $t = \min_{n_i \in M} T_i$ denote the minimum number of packet injections since initialization taken over all the nodes of the network, and let $S = \{n_i \in M | T_i \leq T_j, \forall n_j \in M\}$ denote the set of nodes inside the network that has the minimum number of packet injections since initialization. The relative differences i in the number of packet injections of each of $n \in S$'s neighbors must all be ≥ 0. Since $k > 0$, according to our protocol, every node $n \in S$ is permitted to inject. Lemma 7 then guarantees eventual injections from all of the nodes in S. Hence t, the minimum number of packet injections per node, is guaranteed to eventually increase in value over time. Therefore, T_i is guaranteed to eventually increase in value over time $\forall n_i \in M$. **Q.E.D.**

Since the total number of packet injections per node is guaranteed to be strictly increasing over time, we are assured of the eventual packet injection for each of the individual nodes of the network. Therefore, the above theorem establishes *fairness* in network access among all the nodes. We now prove that the above protocol actually allows us to establish *strong fairness* among the nodes of our network under the consumption assumption. That is, each node that has a packet pending injection will be allowed to inject within a bounded finite period of time. We now proceed to establish this strong fairness result.

Corollary 9 Let M be a finite network as described in Theorem 8 above and satisfies the

consumption assumption, then each node that has a packet pending injection will inject within a bounded period.

Proof. We proceed to prove the stated result by exhibiting such a finite albeit unrealistically pessimistic bound. Let b, n and d be defined as in Theorem 5 of the last section. Let $t = \min T_i$, and $T = \max T_i$, denote the minimum and maximum number of injections per node since initialization among all the nodes of M. Then we have $T \leq t + kd$ according to our synchronization protocol. Let t_i and T_i denote the the respective values of t and T after each routing cycle c_i, $\forall i$. Specifically, let c_0 denote the current routing cycle. Since the values of t and T increases strictly over time, we are guaranteed that there exists in the future, a routing cycle c_m such that $t_m > T_0$ and $t_i \leq T_0$, $\forall i$, $0 < i < m$. We assert that during the cycles c_1 upto c_m, there cannot have been more than $n(2kd+1)$ packet injections into the network. To see this, observe that we have, $T_m - t_m \leq kd$ and, $T_0 - t_0 \leq kd$. Adding them together we have $T_m - t_0 \leq 2kd + t_m - T_0$. But c_m was defined such that $t_m - T_0 = 1$, and hence $T_m - t_0 \leq 2kd + 1$. Since there are a total of n nodes in the network, the assertion follows directly from the *pigeon hole* principle. Now, from Theorem 5, we conclude that there is at least one packet injection within every bnd routing cycles. Therefore the total number of routing cycles m elapsed must be $\leq n(2kd+1)(bnd)$. Since $t_m > T_0$, we are guaranteed of at least one injection per node in our network every m cycles. The above bound on m then establishes the validity of the required strong fairness result.

Q.E.D.

The above theorem suggests that by using the local synchronization protocol and packet injection policy described above together with the introduction of packet delivery guaranteed priorities described in the previous section, we can guarantee progress in every individual node and every individual packet inside a finite network, a very desirable state of affairs.

It is interesting to compare this injection control scheme with our previous token recirculation scheme. As have been pointed out, the token recirculation scheme is inefficient in that a pending node under heavy traffic has to wait for the occurrence of an empty buffer brought back by reply packets across the network. In contrast, in the injection control scheme, a pending node under heavy traffic prohibits further injections of its neighbors, and the area under *curfew* extends gradually outward if congestion persists. Intuitively, this allows the pending node to capture empty buffers starting from its local vicinity, which appears to be more efficient. Another difference of the injection scheme compared to the token recirculation scheme is that it tends to regulate the total network packet population, preventing message latencies from growing excessive under heavy traffic condition. In

contrast, under the token recirculation scheme, more round trip packets are injected into the network precisely when heavy congestion occurs. This further overloads the network resources and increases the message latencies substantially. However, the token recirculation scheme, being more tolerant towards transient local traffic fluctuations, appears to be able to substain higher throughput under light to moderate traffic conditions. The relative merits of the two approaches is a subject that certainly requires further study.

6 Conclusions

We have described informally the basic ideas and motivations behind the adaptive approach for message routing in multicomputer networks. A simple model for the discussion of adaptive routing strategies is presented that allows us to discuss a number of fundamental issues such as communication deadlock, eventual message packet delivery and eventual message packet injection.

An argument based on the desire to be able to exploit alternate routes in order to lessen local network congestions is given which led to the adoption of the packet exchange model. The exchange model renders communication deadlock a non-issue. It allows us to exploit arbitrary alternate routes without having to worry about potential communication deadlock which is difficult and expensive to avoid using more conventional means.

A general packet prioritizing scheme for constructing packet delivery guaranteed routing algorithms for arbitrary network and adaptive control is developed. The priority scheme developed is then shown to be lockout free and under the consumption assumption assures delivery of message packets within a bound period over a finite network.

The problem of assuring eventual message packet injection at pending nodes is examined in detail. The two conditions under which a pending node is allowed to inject its queued packet is discussed. Two different solutions are then presented each corresponds to guaranteeing the eventual occurrence of one of the two desirable conditions. The two solutions are each examined in detail, and a heuristic discussion of the relative merits of the two approaches is given.

The main emphasis of this report is in laying down a basic foundation upon which one can explore various strategies and extensions of the adaptive framework for message routing in multicomputer networks. We have focused on addressing issues which are fundamental to any message communication network supporting reliable concurrent computations and

we did not find any insurmountable problem. Rather, the simplicity of these resolution mechanisms give us hope that adaptive schemes may be found that improve on the already highly evolved oblivious routing schemes.

7 References

[1] Seitz, Charles L., "The Cosmic Cube", *CACM*, 28(1), January 1985, pp. 22-33.

[2] *Submicron Systems Architecture Semiannual Technical Report*, Computer Science Department, Caltech, 5240:TR:87.

[3] Dally, William J., *A VLSI Architecture for Concurrent Data Structures*, Ph.D. Thesis, Computer Science Department, Caltech, 5209:TR:86.

[4] Dally, William J. and Seitz, Charles L., "The torus routing chip", *Distributed Computing*, 1986(1), pp. 187-196.

[5] Borodin, A. and Hopcroft, J., "Routing, Merging, and Sorting on Parallel Models of Computation", *Journal of Computer and System Sciences*, **30**, pp. 130-145 (1985).

[6] Dally, William J. and Seitz, Charles L., "Deadlock-Fee Message Routing in Multiprocessor Interconnection Networks", *IEEE Transactions on Computers*, Vol. C-36, No. 5, pp. 547-553, May 1987.

[7] Merlin, P. and Schweitzer, P., "Deadlock Avoidance in Store-and-Forward Networks – I : Store-and Forward Deadlock", *IEEE Transactions on Communications*, Vol. COM-28, No. 3, pp. 345-354, March 1980.

[8] Dally, William J., "Wire-Efficient VLSI Multiprocessor Communication Networks", *Advanced Research in VLSI, Proceedings of the 1987 Stanford Conference*, MIT Press, pp. 391-415.